CS294-1 On-line Computation & Network Algorithms **Spring 1997**

Lecture 13: March 4

Lecturer: Yair Bartal *Scribe: Andrew Swan*

13.1 The Greedy Steiner Tree Algorithm

Recall that in the on-line Steiner tree problem [IW], the input consists of a graph G and a series of vertices $v_1, v_2, \ldots, v_n$. At each vertex request v_i, the algorithm must compute T_i, a Steiner tree that spans $v_1, v_2, \ldots, v_i$ with the constraint $T_{i+1} \supseteq T_i$. That is, the tree is constructed incrementally at every request. The cost incurred by the algorithm is the cost of the final tree T_n.

The Greedy Steiner Tree algorithm (GST) simply pays the minimum incremental cost at every step. A similar greedy algorithm exists for the on-line spanning tree problem, although it is much simpler since the tree constructed in the on-line spanning tree problem may not include any nodes other than the v_i. Note that since a Steiner tree is also a spanning tree, a bound for the cost of the greedy spanning tree algorithm is also a bound for the cost of GST, although the bound may not be very strong.

The following results are from Imaze and Waxeman:

Theorem 13.1 *The competitive ratio for a greedy Steiner tree algorithm is* $\lceil \log n \rceil$.

Theorem 13.2 *The competitive ratio for any on-line Steiner tree algorithm has a lower bound of* $\frac{1}{2}\lfloor \log n \rfloor$.

As a simple case, consider a line with n equally spaced points. Let the first request be for the leftmost point and the second be for the rightmost point. Then let all further requests be for points in the middle of the remaining regions; the first two iterations are shown in Figure 13.1. The GST algorithm will pay a cost of $n \log n$ while the optimal cost is n for a competitive ratio of $\log n$.

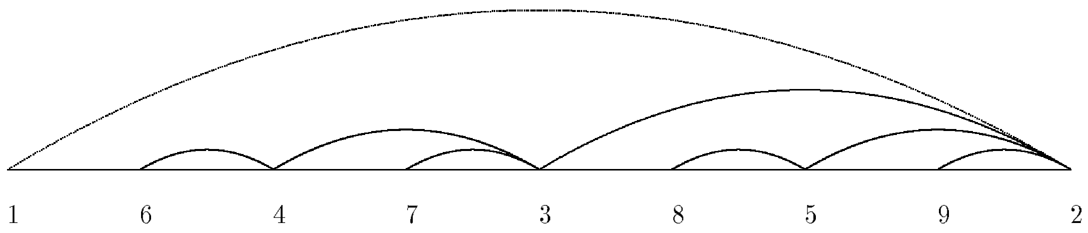

Figure 13.1: The first few requests to the GST algorithm from a sequence for which the tree costs $n \log n$

A similar argument is used to show a general lower bound for the GST algorithm.

Begin by constructing a cycle that includes all the vertices, for example the dotted line in Figure 13.2. If the node u was requested before the node v, then let $w(u, v)$ be the distance between u and v on the tree.

Now, group the vertices in pairs; there are two ways in which the vertices could be paired, call them P_1 and P_2. Define W_1 and W_2 as follows:

$$W_1 = \sum (u,v) \in P_1 w(u,v)$$

$$W_2 = \sum (u,v) \in P_2 w(u,v)$$

If W is the total weight of the cycle and OPT is the total weight of the optimal Steiner tree, then

$$W_1 + W_2 \leq W \leq 2\mathrm{OPT}$$

Now, assume $W_1 \leq W_2$, thus

$$W_1 \leq W/2 \leq \mathrm{OPT}$$

Consider the set constructed by taking from each pair in P_1, the node that was requested later. This set has $\lfloor n/2 \rfloor$ nodes for which the aggregate cost is at most $W_1 < \mathrm{OPT}$. This analysis can be repeated recursively on the remaining nodes. It will be repeated $\lceil \log n \rceil$ times for a total cost of $\mathrm{OPT}\lceil \log n \rceil$ and thus a competitive ratio of $\lceil \log n \rceil$.

The following series of requests show the lower bound. In the graph on the left side of Figure 13.3, let the first two requests be at nodes 1 and 2. The tree that is constructed must go through either the top or the bottom. Assuming it goes through the top, replace the bottom edges with copies of the same graph, generating the second graph in Figure 13.3. If this process is repeated, it is easy to show that the number of nodes in the tree at every step is:

$$n_i = n_{i-1} + 2(4^i) \in O(4^i)$$

while the optimal cost is given by

$$\mathrm{OPT}_i = \mathrm{OPT}_{i-1} + 2^i = L \in \Theta(2^i)$$

and the on-line cost is

$$\mathrm{ON} = L + L/2 + 2(L/4) + 4(L/8) + \cdots = L\lceil \log n \rceil$$

and thus the competitive ratio c is $\Omega(\lceil \log n \rceil)$.

Some Open Problems

1. Is there an algorithm for the on-line Steiner tree problem that does better than $\lceil \log n \rceil$ on the Euclidean plane? A lower bound of $\Omega(\frac{\log n}{\log \log n})$ was shown in [AA].
2. Is there a graph such that GST is not within a constant factor of the best competitive ratio?
3. Steiner tree algorithms for particular graphs, generalizations, etc.

13.2 A Deterministic File Replication Algorithm

This algorithm was originally presented in [ABF1].

The algorithm maintains a list L of read requests. Upon a read request at node r, r is inserted in to L and then the algorithm finds the smallest sphere around r that contains D read requests from L. Let k be the radius of that sphere. If no copy of the file exists within a sphere of radius λk, (where λ is some paramter for the algorithm) the file is replicated to r and all the requests within the first sphere are removed from L. Writes requests are handled by updating all existing copies with no replication or deletion. However, to limit the overhead from write requests, all but one copy will be deleted after D write requests. If these D requests were at nodes $w_1, w_2, \ldots, w_D$, then a copy is maintained at node m where m minimizes some function (e.g., the distance from m to all w_i $\sum d(m, w_i)$. The write request counter is then reset and the algorithm proceeds.

Theorem 13.3 *This algorithm is $O(\log n)$ competitive.*

The proof is based on a variant of the GST proof. It looks at the optimal treeand considers equal neighborhoods that cover the whole tree. A potential function is defined that "credits" the adversary when it has a copy of the file near a request but ALG does not. This analysis is repeated for $\lceil \log n \rceil$ levels of covering.

13.3 Competitive Distributed Algorithms

Distributed algorithms for the file replication problem and other problems impose an additional constraint on the amount of memory at every node. Distributed algorithms for many of the problems discussed so far exist although they are generally not as good as the non-distributed versions.

Besides the obvious partial information about other nodes in the network, issues in the design of distributed algorithms include congestion and concurrency. Standard distributed algorithms exchange control messages. A simple technique on a uniform network is to elect a single "leader" node and run a centralized algorithm at that node and communicate from the leader to all the other nodes. It is often possible to do much better than that however. Finally, distributed algorithm design is somewhat simplified on trees since it is clear how control messages should be exchanged (since there is only one path between any two nodes).

References

[AA92] N. Alon and Y. Azar. On-line Steiner Trees in the Euclidean Plane. In *Proc. 8th ACM Symp. on Computational Geometry*, pages 337-343, 1992.

[ABF93a] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive Distributed File Allocation. In *Proc. of the 25th Ann. ACM Symp. on Theory of Computing*, pages 164-173, May 1993.

[IW91] M. Imaze and B.M. Waxman. Dynamic Steiner Tree Problem. In *SIAM Journal on Discrete Mathematics*, 4(3):369-384, August 1991.

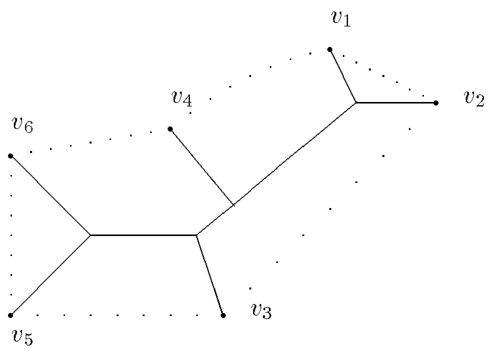

Figure 13.2: A simple Steiner tree with the vertices connected by a cycle

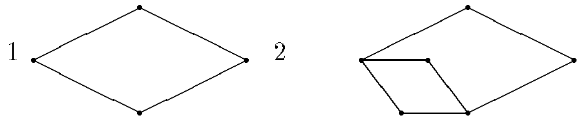

Figure 13.3: Graphs with a lower bound for the Steiner tree problem of $\Theta(\lceil \log n \rceil)$